A4-Q3Q4: DCT and JPEG Compression

In [1]:
```
import numpy as np
from numpy.fft import fft, ifft, fft2, ifft2
import matplotlib.pyplot as plt
```

Q3: Discrete Cosine Transform

Some helper functions

In [2]:
```
def EvenExtension(f):
    '''
     fe = EvenExtension(f)

     Performs an even extension on the array f.

     Input:
       f is a 2D array

     Output:
       fe is the even extension of f

     If f has dimensions NxM, then fe has dimensions
        (2*N-2)x(2*M-2)
     and fe[n,j]=fe[-n,j] for n=0,...,N-1
     and fe[n,j]=fe[n,-j] for j=0,...,M-1

     For example, if f is 5x4, then fe has dimensions 8x6.

     IEvenExtension is the inverse of EvenExtension, so that
        IEvenExtension(EvenExtension(f)) == f
     for any matrix f.

    '''

    fe = np.concatenate((f,np.fliplr(f[:,1:-1])), axis=1)
    fe = np.concatenate((fe, np.flipud(fe[1:-1,:])), axis=0)
    return fe

def IEvenExtension(fe):
    '''
     f = IEvenExtension(fe)

     Reverses the action of an even extension.

     Input:
       fe is a 2D array, assumed to contain an even extension

     Output:
       f is the sub-array that was used to generate the extension

     If fe has dimensions KxL, then f has dimensions
        ceil((K+1)/2) x ceil((L+1)/2)
     For example, if fe is 8x6, then f is 5x4.

     IEvenExtension is the inverse of EvenExtension, so that
```

```
       IEvenExtension(EvenExtension(f)) == f
     for any matrix f.

    '''

    e_dims = np.array(np.shape(fe))
    dims = np.ceil((e_dims+1.)/2)
    dims = np.array(dims, dtype=int)
    f = fe[:dims[0], :dims[1]]
    return f
```

In [3]:

```
# Define a simple 2-D array to play with
f = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]], dtype=float)
print(f)
```

```
[[ 1.  2.  3.  4.]
 [ 5.  6.  7.  8.]
 [ 9. 10. 11. 12.]]
```

In [4]:

```
# Even extension
fe = EvenExtension(f)
print(fe)
```

```
[[ 1.  2.  3.  4.  3.  2.]
 [ 5.  6.  7.  8.  7.  6.]
 [ 9. 10. 11. 12. 11. 10.]
 [ 5.  6.  7.  8.  7.  6.]]
```

In [5]:

```
# Check that it's even, if you don't believe me
n = np.random.randint(np.shape(f)[0])
j = np.random.randint(np.shape(f)[1])
print((n,j))
print(fe[n,j])
print(fe[-n,-j])
```

```
(1, 0)
5.0
5.0
```

In [6]:

```
# Inverse even extension
g = IEvenExtension(fe)
print(g)
```

```
[[ 1.  2.  3.  4.]
 [ 5.  6.  7.  8.]
 [ 9. 10. 11. 12.]]
```

myDCT

In [7]:

```
def myDCT(f):
    '''
     Fdct = myDCT(f)

     Computes the 2-D Discrete Cosine Transform of input image f.
     It uses an even extension of f, along with the 2D-DFT.
     This function is the inverse of myIDCT.

     Input:
      f is a 2-D array of real values

     Output:
```

```
      Fdct is a real-valued array the same size as f
    '''

    fe = EvenExtension(f)
    F = fft2(fe)
    return IEvenExtension(F).real
```

myIDCT

In [8]:
```
def myIDCT(Fdct):
    '''
     f = myIDCT(Fdct)

     Computes the 2-D Inverse Discrete Cosine Transform (IDCT) of input
     array Fdct. It uses an even extension of Fdct, along with the 2D-IDFT.
     This function is the inverse of myDCT.

     Input:
      Fdct is a 2-D array of real values

     Output:
      f is a real-valued array the same size as Fdct
    '''

    Fe = EvenExtension(Fdct)
    f = ifft2(Fe)
    return IEvenExtension(f).real
```

In [ ]:
```

```

Q4: JPEG Compression

In [9]:
```
# A couple functions to help you
def NumPixels(f):
    '''
     n = NumPixels(f) returns the total number of elements in the array f.

     For example,
       NumPixels( np.ones((5,4)) )
     returns the value 20.
    '''
    return np.prod(np.shape(f))

def Show(g, title=''):
    '''
     Show(g, title='')

     Displays the image g as a graylevel image with intensities
     clipped to the range [0,255].
    '''
    plt.imshow(np.clip(g, a_min=0, a_max=255), cmap='gray');
    plt.axis('off');
    plt.title(title);
```

myJPEGCompress

In [10]:

```
def myJPEGCompress(f, T, D):
    '''
     G = myJPEGCompress(f, T, D)

     Input
        f is the input image, a 2D array of real numbers
        T is the tile size to break the input image into
        D is the size of the block of Fourier coefficients to keep
          (Bigger values of D result in less loss, but less compression)

     Output
        G is the compressed encoding of the image

     Example: If f is 120x120, then

        G = myJPEGCompress(f, 10, 4)

     would return an array (G) of size 48x48.
    '''
    i_max = len(f)
    j_max = len(f[0])
    ff = np.zeros([T, T])
    G = np.zeros([int(i_max*D/T), int(j_max*D/T)])

    row = 0
    for i in range(0, i_max, T):
        col = 0
        for j in range(0, j_max, T):
            # for each tile
            tmp = f[i:i+T]
            for h in range(T):
                ff[h] = tmp[h][j:j+T]
            FF = myDCT(ff)
            # now FF contains the DCT of the current T*T tile
            DD = np.zeros([D, D])
            for h in range(D):
                DD[h] = FF[h][:D]
            # now DD contains the D*D sub-array of FF
            for ii in range(D):
                for jj in range(D):
                    G[row*D+ii][col*D+jj] = DD[ii][jj]
            col = col + 1
        row = row + 1
    return G
```

myJPEGDecompress

In [11]:

```
def myJPEGDecompress(G, T, D):
    '''
     f = myJPEGDecompress(G, T, D)

     Input
        G is the compressed encoding, a 2D array of real numbers
        T is the tile size for reassembling the decompressed image
        D is the size of the blocks of Fourier coefficients that were
          kept when the image was compressed
          (Bigger values of D result in less loss, but less compression)

     Output
```

```
        f is the decompressed, reconstructed image

     Example: If G is 48x48, then

        f = myJPEGDecompress(G, 10, 4);

     would return an array (f) of size 120x120.
    '''

    i_max = len(G)
    j_max = len(G[0])
    f = np.zeros([int(i_max*T/D), int(j_max*T/D)])

    row = 0
    for i in range(0, i_max, D):
        col = 0
        for j in range(0, j_max, D):
            # for each tile
            gg = np.zeros([T, T]) # initialize with all zeros
            for ii in range(D):
                for jj in range(D):
                    gg[ii][jj] = G[D*row+ii][D*col+jj] # fill in the D*D portion
            GG = myIDCT(gg)
            # now GG contains the IDCT of the current T*T tile
            for ii in range(T):
                for jj in range(T):
                    f[row*T+ii][col*T+jj] = GG[ii][jj]
            col = col + 1
        row = row + 1

    return f
```

Demonstrate Compression

In [12]:

```
f = plt.imread('Jinan.jpg')[:,:,0]
Show(f)
```

In [13]:

```
G1 = myJPEGCompress(f, 25, 12)
f1 = myJPEGDecompress(G1, 25, 12)
rate = round(NumPixels(f)/NumPixels(G1), 2)
Show(f1)
plt.title("Jinan.jpg (Compression Ratio = " + str(rate) + ":1)")
```

Out[13]: `Text(0.5, 1.0, 'Jinan.jpg (Compression Ratio = 4.34:1)')`

Jinan.jpg (Compression Ratio = 4.34:1)

In [14]:
```
G2 = myJPEGCompress(f, 25, 8)
f2 = myJPEGDecompress(G2, 25, 8)
rate = round(NumPixels(f)/NumPixels(G2), 2)
Show(f2)
plt.title("Jinan.jpg (Compression Ratio = " + str(rate) + ":1)")
```

Out[14]:
```
Text(0.5, 1.0, 'Jinan.jpg (Compression Ratio = 9.77:1)')
```

Jinan.jpg (Compression Ratio = 9.77:1)

In [15]:
```
G3 = myJPEGCompress(f, 25, 5)
f3 = myJPEGDecompress(G3, 25, 5)
rate = round(NumPixels(f)/NumPixels(G3), 2)
Show(f3)
plt.title("Jinan.jpg (Compression Ratio = " + str(rate) + ":1)")
```

Out[15]:
```
Text(0.5, 1.0, 'Jinan.jpg (Compression Ratio = 25.0:1)')
```

Jinan.jpg (Compression Ratio = 25.0:1)